PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: Sachin Mullick, et al.	Technology Center: 2100
Serial No.: 10/668.467 Confirm: 2942	Group Art Unit: 2161
Filed : 09/23/2003	Examiner: Bibbee, Jared M
For: Multi-Threaded Write Interface and Methods for Increasing the Single File Read and Write Throughput of a File Server	Atty. Dkt. No.: 10830.0100.NPUS00

APPEAL BRIEF TO THE BOARD OF PATENT APPEALS AND INTERFERENCES

Commissioner for Patents
PO Box 1450
Alexandria, Virginia 22313-1450

Sir:

This Appeal Brief is in support of Appellants' Notice of Appeal filed Aug. 22, 2008 from the final Official Action dated May 23, 2008. Please deduct any deficiency in any required fee from EMC Corporation Deposit Account No. 05-0889.

I. REAL PARTY IN INTEREST

The real party in interest is EMC Corporation, by virtue of assignments recorded at Reel 014554 Frame 0097 and 014880 Frame 0711.

II. RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences.

III. STATUS OF THE CLAIMS

Claims 1-73 have been presented for examination.

Claims 29-31, 55-57 have been canceled.

Claims 1-28, 32-54 and 58-73 have been finally rejected.

Claims 1-28, 32-54, 58-65, 67-71, and 73 are being appealed.

IV. STATUS OF AMENDMENTS

No amendment was filed after the final Official Action of May 23, 2008.

V. SUMMARY OF CLAIMED SUBJECT MATTER

The invention of appellants' independent claim 1 is a method of operating a network file server (21 in appellants' FIG. 1; appellants' specification, page 11, lines 4-8) for providing clients (23, 24, 25 in FIG. 1; page 1 lines 3-4) with concurrent write access (page 15 lines 17-19) to a file (FIG. 14; page 29, line 22 to page 30, line 3). (Appellants' specification, page 3, lines 19-21.) The method includes the network file server responding to a concurrent write request from a client by obtaining a lock for the file (step 101 in FIG. 11), and then preallocating a metadata block for the file (step 102 in FIG. 11), and then releasing the lock for the file (step 103 in FIG. 11); and then asynchronously writing to the file (step 104 in FIG. 11); and then obtaining the lock for the file (step 106 in FIG. 11); and then committing the metadata block to the file (step 107 in FIG. 11); and then releasing the lock for the file (step 108 in FIG. 11). (Appellants' specification, page 3 line 21 to page 4 line 2; page 27 lines 3-23.) Appellants' FIG. 11 is reproduced below.

The invention of appellants' independent claim 13 is a method of operating a network file server (21 in appellants' FIG. 1; appellants' specification, page 11, lines 4-8) for providing clients (23, 24, 25 in FIG. 1; page 1 lines 3-4) with concurrent write access (page 15 lines 17-19) to a file (FIG. 14; page 29, line 22 to page 30, line 3). (Appellants' specification, page 4, lines 3-4.) The method includes the network file server responding to a concurrent write request from a client by preallocating a block for the file (step 102 in FIG. 11); and then asynchronously writing

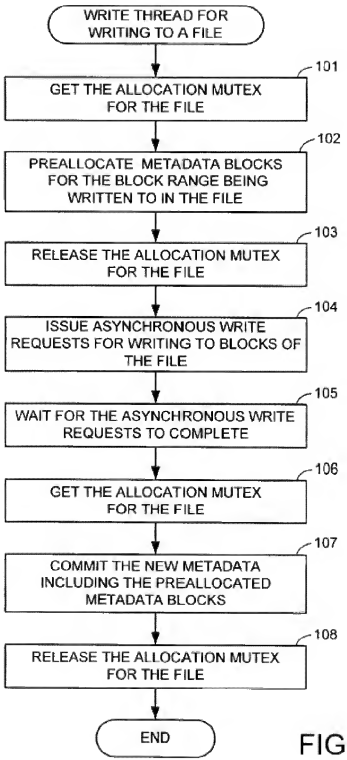
WRITE THREAD FOR WRITING TO A FILE
101
GET THE ALLOCATION MUTEX FOR THE FILE
102
PREALLOCATE METADATA BLOCKS FOR THE BLOCK RANGE BEING WRITTEN TO IN THE FILE
103
RELEASE THE ALLOCATION MUTEX FOR THE FILE
104
ISSUE ASYNCHRONOUS WRITE REQUESTS FOR WRITING TO BLOCKS OF THE FILE
105
WAIT FOR THE ASYNCHRONOUS WRITE REQUESTS TO COMPLETE
106
GET THE ALLOCATION MUTEX FOR THE FILE
107
COMMIT THE NEW METADATA INCLUDING THE PREALLOCATED METADATA BLOCKS
108
RELEASE THE ALLOCATION MUTEX FOR THE FILE
END

FIG. 11

to the file (step 104 in FIG. 11); and then committing the block to the file (step 107 in FIG. 11). (Appellants' specification, page 4, lines 4-7; page 27 lines 4-12 and 14-20.) The asynchronous writing to the file includes a partial write to a new block (123 in FIG. 13) that has been copied at least in part from an original block (121 in FIG. 13) of the file. (Appellants' specification, page 4, lines 7-9; page 28 line 22 to page 29 line 7.) The method includes checking a partial block conflict queue (73 in FIG. 4; page 14 lines 19-22; page 15 lines 20-22) for a conflict with a concurrent write to the new block (step 151 in FIG. 17; page 34 lines 7-10), and upon finding an indication of a conflict with a concurrent write to the new block, waiting until resolution of the conflict with the concurrent write to the new block (step 156 in FIG. 17; page 34 line 24 to page 35 line 2), and then performing the partial write to the new block (step 157 in FIG. 17; page 35 lines 2-5). (Appellants' specification, page 4, lines 7-13.) Appellants' FIGS. 4, 13 and 17 are reproduced below.

The invention of appellants' independent 15 is a method of operating a network file server (21 in appellants' FIG. 1; appellants' specification, page 11, lines 4-8) for providing clients (23, 24, 25 in FIG. 1; page 1 lines 3-4) with concurrent write access (page 15 lines 17-19) to a file (FIG. 14; page 29, line 22 to page 30, line 3). (Appellants' specification, page 4, lines 14-15.) The method includes the network file server responding to a concurrent write request from a client by preallocating a metadata block for the file (step 102 in FIG. 11), and then asynchronously writing to the file (step 104 in FIG. 11), and then committing the metadata block to the file (step 107 in FIG. 11). (Appellants' specification, page 4, lines 15-18; page 27 lines 4-

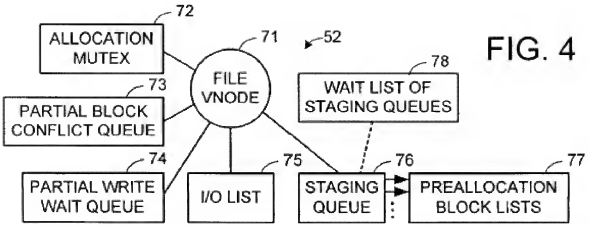
72
ALLOCATION MUTEX
71
52
FILE VNODE
78
WAIT LIST OF STAGING QUEUES
73
PARTIAL BLOCK CONFLICT QUEUE
74
PARTIAL WRITE WAIT QUEUE
75
I/O LIST
76
STAGING QUEUE
77
PREALLOCATION BLOCK LISTS

FIG. 4

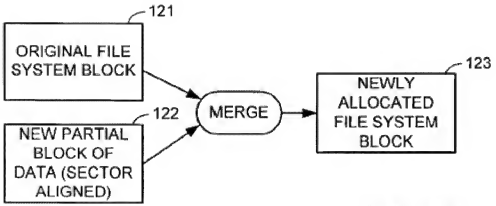
121
ORIGINAL FILE SYSTEM BLOCK
122
NEW PARTIAL BLOCK OF DATA (SECTOR ALIGNED)
MERGE
123
NEWLY ALLOCATED FILE SYSTEM BLOCK

FIG. 13

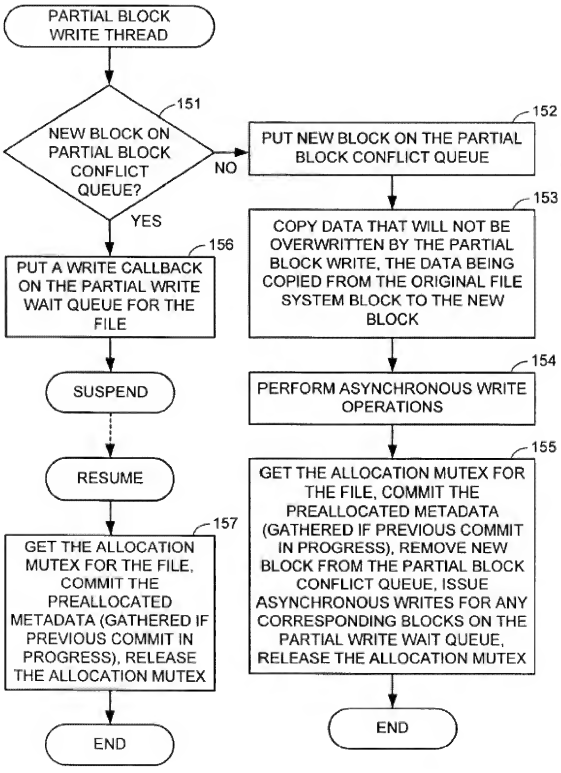
PARTIAL BLOCK WRITE THREAD
151
NEW BLOCK ON PARTIAL BLOCK CONFLICT QUEUE?
NO
YES
152
PUT NEW BLOCK ON THE PARTIAL BLOCK CONFLICT QUEUE
153
COPY DATA THAT WILL NOT BE OVERWRITTEN BY THE PARTIAL BLOCK WRITE, THE DATA BEING COPIED FROM THE ORIGINAL FILE SYSTEM BLOCK TO THE NEW BLOCK
154
PERFORM ASYNCHRONOUS WRITE OPERATIONS
155
GET THE ALLOCATION MUTEX FOR THE FILE, COMMIT THE PREALLOCATED METADATA (GATHERED IF PREVIOUS COMMIT IN PROGRESS), REMOVE NEW BLOCK FROM THE PARTIAL BLOCK CONFLICT QUEUE, ISSUE ASYNCHRONOUS WRITES FOR ANY CORRESPONDING BLOCKS ON THE PARTIAL WRITE WAIT QUEUE, RELEASE THE ALLOCATION MUTEX
END
156
PUT A WRITE CALLBACK ON THE PARTIAL WRITE WAIT QUEUE FOR THE FILE
SUSPEND
RESUME
157
GET THE ALLOCATION MUTEX FOR THE FILE, COMMIT THE PREALLOCATED METADATA (GATHERED IF PREVIOUS COMMIT IN PROGRESS), RELEASE THE ALLOCATION MUTEX
END

FIG. 17

12 and 14-20.) The method further includes gathering together preallocated metadata blocks for a plurality of client write requests to the file (step 117 in FIG. 12), and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining a lock for the file (step 106 in FIG. 11), committing the gathered preallocated metadata blocks for the plurality of client write requests to the file (step 107 in FIG. 11; step 118 in FIG. 12), and then releasing the lock for the file (step 108 in FIG. 11). (Appellants' specification, page 4, lines 18-23; page 27 lines 14-23; page 28, lines 9-21.) Appellants' FIG. 12 is reproduced below.

The invention of appellants' independent 25 is a method of operating a network file server (21 in appellants' FIG. 1; appellants' specification, page 11, lines 4-8) for providing clients (23, 24, 25 in FIG. 1; page 1 lines 3-4) with concurrent read and write access (page 15 lines 17-19) to a file (FIG. 14; page 29, line 22 to page 30, line 3). The method includes the network file server responding to a concurrent write request from a client by preallocating a metadata block for the file (step 102 in FIG. 11), and then asynchronously writing to the file (step 104 in FIG. 11), and then committing the metadata block to the file (step 107 in FIG. 11). (Appellants' specification, page 27 lines 4-12 and 14-20.) The network file server includes disk storage (29 in FIGS. 1 and 2; page 11 lines 3-6; page 12 lines 19-21) containing a file system (54 in FIG. 2 and FIG. 3; page 12 lines 6-9 and 19-21; page 13 lines 11-16), and a file system cache (51 in FIG. 2 and FIG. 3; page 12 lines 21-23; page 13 lines 8-16) storing data of blocks (134, 135, 138, 139 in FIG. 14; page 29, line 22 to page 30, line 3) of the file. The method includes the network file server responding to concurrent write requests by writing new data for specified

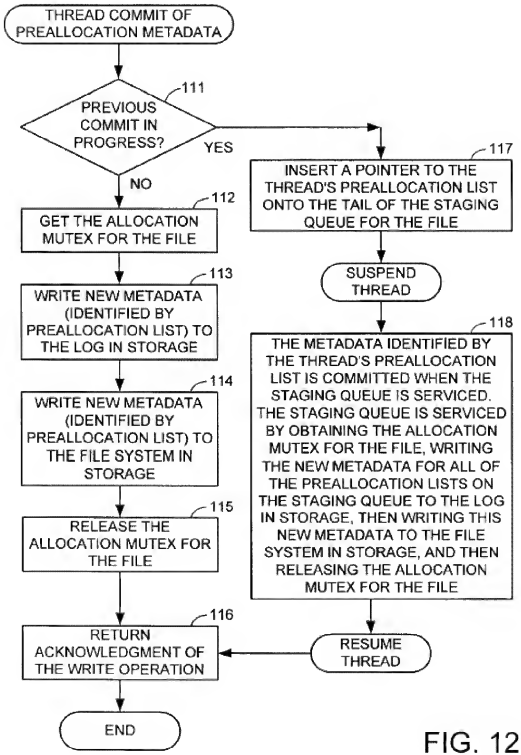
THREAD COMMIT OF PREALLOCATION METADATA
111
PREVIOUS COMMIT IN PROGRESS?
YES
NO
112
GET THE ALLOCATION MUTEX FOR THE FILE
113
WRITE NEW METADATA (IDENTIFIED BY PREALLOCATION LIST) TO THE LOG IN STORAGE
114
WRITE NEW METADATA (IDENTIFIED BY PREALLOCATION LIST) TO THE FILE SYSTEM IN STORAGE
115
RELEASE THE ALLOCATION MUTEX FOR THE FILE
116
RETURN ACKNOWLEDGMENT OF THE WRITE OPERATION
END
117
INSERT A POINTER TO THE THREAD'S PREALLOCATION LIST ONTO THE TAIL OF THE STAGING QUEUE FOR THE FILE
SUSPEND THREAD
118
THE METADATA IDENTIFIED BY THE THREAD'S PREALLOCATION LIST IS COMMITTED WHEN THE STAGING QUEUE IS SERVICED. THE STAGING QUEUE IS SERVICED BY OBTAINING THE ALLOCATION MUTEX FOR THE FILE, WRITING THE NEW METADATA FOR ALL OF THE PREALLOCATION LISTS ON THE STAGING QUEUE TO THE LOG IN STORAGE, THEN WRITING THIS NEW METADATA TO THE FILE SYSTEM IN STORAGE, AND THEN RELEASING THE ALLOCATION MUTEX FOR THE FILE
RESUME THREAD

FIG. 12

blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache. (Steps 515 and 516 in FIG. 10; page 23 lines 19-23 and page 24 lines 7-15.) The method further includes the network file server responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing to the file system cache a file block that has become stale. (Steps 92, 97, 513, 98 in FIG. 10; page 23 lines 5-17 page 24 lines 16-22.) Appellants' FIG. 10 is reproduced below.

The invention of appellants' independent claim 32 is a method of operating a network file server (21 in appellants' FIG. 1; appellants' specification, page 11, lines 4-8) for providing clients (23, 24, 25 in FIG. 1; page 1 lines 3-4) with concurrent write access (page 15 lines 17-19) to a file (FIG. 14; page 29, line 22 to page 30, line 3). (Appellants' specification, page 5, lines 1-2.) The method includes the network file server responding to a concurrent write request from a client by executing a write thread (FIG. 11). (Appellants' specification, page 4, lines 3-4.). Execution of the write thread includes obtaining an allocation mutex for the file (step 101 in FIG. 11), and then preallocating new metadata blocks that need to be allocated for writing to the file (step 102 in FIG. 11), and then releasing the allocation mutex for the file (step 103 in FIG. 11); and then issuing asynchronous write requests for writing to the file (step 104 in FIG. 11), waiting for callbacks indicating completion of the asynchronous write requests (step 105 in FIG. 11), and

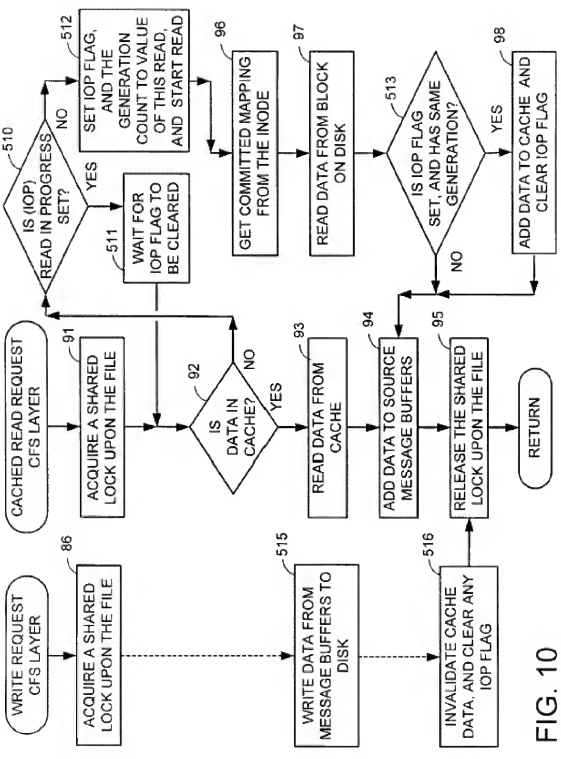
WRITE REQUEST CFS LAYER
86
ACQUIRE A SHARED LOCK UPON THE FILE
515
WRITE DATA FROM MESSAGE BUFFERS TO DISK
516
INVALIDATE CACHE DATA, AND CLEAR ANY IOP FLAG
CACHED READ REQUEST CFS LAYER
91
ACQUIRE A SHARED LOCK UPON THE FILE
92
IS DATA IN CACHE?
NO
YES
93
READ DATA FROM CACHE
94
ADD DATA TO SOURCE MESSAGE BUFFERS
95
RELEASE THE SHARED LOCK UPON THE FILE
RETURN
510
IS (IOP) READ IN PROGRESS SET?
YES
511
WAIT FOR IOP FLAG TO BE CLEARED
NO
512
SET IOP FLAG, AND THE GENERATION COUNT TO VALUE OF THIS READ, AND START READ
96
GET COMMITTED MAPPING FROM THE INODE
97
READ DATA FROM BLOCK ON DISK
513
IS IOP FLAG SET, AND HAS SAME GENERATION?
NO
YES
98
ADD DATA TO CACHE AND CLEAR IOP FLAG

FIG. 10

then obtaining the allocation mutex for the file (step 106 in FIG. 11), and then committing the preallocated metadata blocks (step 107 in FIG. 11), and then releasing the allocation mutex for the file (step 108 in FIG. 11). (Appellants' specification, page 4, lines 4-10; page 27 lines 3-23.)

The invention of appellants' independent claim 33 is a network file server (21 in appellants' FIG. 1; appellants' specification, page 11, lines 4-8). (Appellants' specification, page 5, line 11.) The network file server includes storage (29 in FIGS. 1 and 2; page 11 lines 3-6; page 12 lines 19-21) for storing a file (FIG. 14; page 29, line 22 to page 30, line 3), and at least one processor (26, 27, 28 in FIG. 1; page 11 lines 4-6) coupled to the storage for providing clients (23, 24, 25 in FIG. 1; page 1 lines 3-4) with concurrent write access (page 15 lines 17-19) to the file. (Appellants' specification, page 5, lines 12-13.) The network file server is programmed for responding to a concurrent write request from a client by obtaining a lock for the file (step 101 in FIG. 11), and then preallocating a metadata block for the file (step 102 in FIG. 11), and then releasing the lock for the file (step 103 in FIG. 11), and then asynchronously writing to the file (step 104 in FIG. 11), and then obtaining the lock for the file (step 106 in FIG. 11), and then committing the metadata block to the file (step 107 in FIG. 11), and then releasing the lock for the file (step 108 in FIG. 11). (Appellants' specification, page 5, lines 13-18; page 27 lines 3-23.)

The invention of appellants' independent claim 47 is a network file server (21 in appellants' FIG. 1; appellants' specification, page 11, lines 4-8). (Appellants' specification, page

5, line 19.) The network file server includes storage (29 in FIGS. 1 and 2; page 11 lines 3-6; page 12 lines 19-21) for storing a file (FIG. 14; page 29, line 22 to page 30, line 3), and at least one processor (26, 27, 28 in FIG. 1; page 11 lines 4-6) coupled to the storage for providing clients (23, 24, 25 in FIG. 1; page 1 lines 3-4) with concurrent write access (page 15 lines 17-19) to the file. (Appellants' specification, page 5, lines 20-21.) The network file server is programmed for responding to a concurrent write request from a client by preallocating a block for the file (step 102 in FIG. 11); and then asynchronously writing to the file (step 104 in FIG. 11); and then committing the block to the file (step 107 in FIG. 11). (Appellants' specification, page 5, line 21 to page 6, line 1; page 27 lines 4-12 and 14-20.) The network file server includes a partial block conflict queue (73 in FIG. 4; page 14 lines 19-22; page 15 lines 20-22) for indicating a concurrent write to a new block that is being copied at least in part from an original block of the file. (Appellants' specification, page 6, lines 1-3.) The network file server is programmed for responding to a client request for a partial write to the new block by checking the partial block conflict queue for a conflict (step 151 in FIG. 17; page 34 lines 7-10), and upon finding an indication of a conflict, waiting until resolution of the conflict with the concurrent write to the new block of the file (step 156 in FIG. 17; page 34 line 24 to page 35 line 2), and then performing the partial write to the new block of the file (step 157 in FIG. 17; page 35 lines 2-5). (Appellants' specification, page 6, lines 3-7.)

The invention of appellants' independent claim 49 is a network file server (21 in appellants' FIG. 1; appellants' specification, page 11, lines 4-8). (Appellants' specification, page

6, line 8.) The network file server includes storage (29 in FIGS. 1 and 2; page 11 lines 3-6; page 12 lines 19-21) for storing a file (FIG. 14; page 29, line 22 to page 30, line 3), and at least one processor (26, 27, 28 in FIG. 1; page 11 lines 4-6) coupled to the storage for providing clients (23, 24, 25 in FIG. 1; page 1 lines 3-4) with concurrent write access (page 15 lines 17-19) to the file. (Appellants' specification, page 6, lines 9-10.) The network file server is programmed for responding to a concurrent write request from a client by preallocating a metadata block for the file (step 102 in FIG. 11), and then asynchronously writing to the file (step 104 in FIG. 11), and then committing the metadata block to the file (step 107 in FIG. 11). (Appellants' specification, page 6, lines 10-13; page 27 lines 4-12 and 14-20.) The network file server is programmed for gathering together preallocated metadata blocks for a plurality of client write requests to the file (step 117 in FIG. 12), and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining a lock for the file (step 106 in FIG. 11), committing the gathered preallocated metadata blocks for the plurality of client write requests to the file (step 107 in FIG. 11; step 118 in FIG. 12), and then releasing the lock for the file (step 108 in FIG. 11). (Appellants' specification, page 6, lines 13-18; page 27 lines 14-23; page 28, lines 9-21.)

The invention of appellants' independent 51 is a network file server (21 in appellants' FIG. 1; appellants' specification, page 11, lines 4-8). The network file server includes disk storage (29 in FIGS. 1 and 2; page 11 lines 3-6; page 12 lines 19-21) containing a file system (54 in FIG. 2 and FIG. 3), and a file system cache (51 in FIG. 2 and FIG. 3; page 12 lines 21-23;

page 13 lines 8-16) storing data of blocks of a file (FIG. 14; page 29, line 22 to page 30, line 3) in the file system. (Appellants' specification, page 6, lines 20-22.) The network file server is programmed for responding to a concurrent write request from a client (23, 24, 25 in FIG. 1; page 1 lines 3-4) by preallocating a metadata block for the file (step 102 in FIG. 11); and then asynchronously writing to the file (step 104 in FIG. 11), and then committing the metadata block to the file (step 107 in FIG. 11). (Appellants' specification, page 27 lines 4-12 and 14-20.) The network file server is further programmed for responding to concurrent write requests by writing new data for specified blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache. (Steps 515 and 516 in FIG. 10; page 23 lines 19-23 and page 24 lines 7-15.) The network file server is programmed for responding to concurrent read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing to the file system cache a file block that has become stale. (Steps 92, 97, 513, 98 in FIG. 10; page 23 lines 5-17 page 24 lines 16-22.)

The invention of appellants' independent claim 58 is a network file server (21 in appellants' FIG. 1; appellants' specification, page 11, lines 4-8). (Appellants' specification, page 6, lines 19-20.) The network file server includes storage (29 in FIGS. 1 and 2; page 11 lines 3-6; page 12 lines 19-21) for storing a file (FIG. 14; page 29, line 22 to page 30, line 3), and at least

one processor (26, 27, 28 in FIG. 1; page 11 lines 4-6) coupled to the storage for providing clients (23, 24, 25 in FIG. 1; page 1 lines 3-4) with concurrent write access (page 15 lines 17-19) to the file. (Appellants' specification, page 6, lines 20-22.) The network file server is programmed with a write thread (FIG. 11) for responding to a concurrent write request from a client by obtaining an allocation mutex for the file (step 101 in FIG. 11), and then preallocating new metadata blocks that need to be allocated for writing to the file (step 102 in FIG. 11), and then releasing the allocation mutex for the file (step 103 in FIG. 11), and then issuing asynchronous write requests for writing to the file (step 101 in FIG. 11), waiting for callbacks indicating completion of the asynchronous write requests (step 101 in FIG. 11), and then obtaining the allocation mutex for the file (step 106 in FIG. 11); and then committing the preallocated metadata blocks (step 107 in FIG. 11), and then releasing the allocation mutex for the file (step 108 in FIG. 11). (Appellants' specification, page 6, line 22 to page 7, line 6; page 27 lines 3-23.)

The invention of appellants' independent claim 61 is a network file server (21 in appellants' FIG. 1; appellants' specification, page 11, lines 4-8). (Appellants' specification, page 7, line 7.) The network file server includes storage (29 in FIGS. 1 and 2; page 11 lines 3-6; page 12 lines 19-21) for storing a file (FIG. 14; page 29, line 22 to page 30, line 3), and at least one processor (26, 27, 28 in FIG. 1; page 11 lines 4-6) coupled to the storage for providing clients (23, 24, 25 in FIG. 1; page 1 lines 3-4) with concurrent write access (page 15 lines 17-19) to the file. (Appellants' specification, page 7, lines 8-9.) The network file server is programmed for

responding to a concurrent write request from a client by preallocating a block for writing to the file (step 102 in FIG. 11), asynchronously writing to the file (step 104 in FIG. 11), and then committing the preallocated block (step 107 in FIG. 11). (Appellants' specification, page 7, lines 9-12; page 27 lines 4-12 and 14-20.) The network file server also includes an uncached write interface (63 in FIG. 3), a file system cache (51 in FIG. 2 and FIG. 3), and a cached read-write interface (61 in FIG. 3). (Appellants' specification, page 7, lines 12-13; page 13 line 8 to page 14 line 17.) The uncached write interface bypasses the file system cache for sector-aligned write operations (FIG. 3; step 85 in FIG. 5), and the network file server is programmed to invalidate cache blocks in the file system cache including sectors being written to by the uncached write interface (FIG. 6, steps 88 and 89). (Appellants' specification, page 7, lines 13-16; page 13 lines 17-22; page 17 lines 12-15; page 18 lines 8-14.) Appellants' FIGS. 3, 5, and 6 are reproduced below.

It is respectfully submitted that none of appellants' claims contain any "means plus function" or "step plus function" as permitted by 35 U.S.C. 112, sixth paragraph.

The invention of appellants' dependent claims 3 and 35 further includes copying data from an original indirect block of the file (121 in FIG. 13; 136 in FIG. 14 and FIG. 15) to the

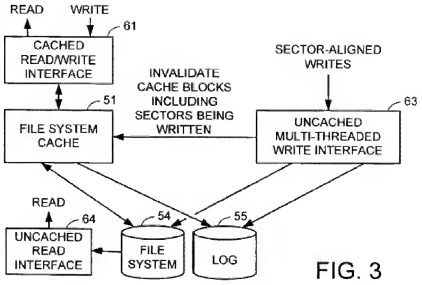
READ
WRITE
61
CACHED READ/WRITE INTERFACE
SECTOR-ALIGNED WRITES
INVALIDATE CACHE BLOCKS INCLUDING SECTORS BEING WRITTEN
51
FILE SYSTEM CACHE
63
UNCACHED MULTI-THREADED WRITE INTERFACE
READ
64
54
55
UNCACHED READ INTERFACE
FILE SYSTEM
LOG

FIG. 3

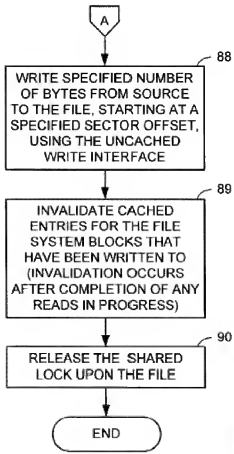
A
88
WRITE SPECIFIED NUMBER OF BYTES FROM SOURCE TO THE FILE, STARTING AT A SPECIFIED SECTOR OFFSET, USING THE UNCACHED WRITE INTERFACE
89
INVALIDATE CACHED ENTRIES FOR THE FILE SYSTEM BLOCKS THAT HAVE BEEN WRITTEN TO (INVALIDATION OCCURS AFTER COMPLETION OF ANY READS IN PROGRESS)
90
RELEASE THE SHARED LOCK UPON THE FILE
END

FIG. 6

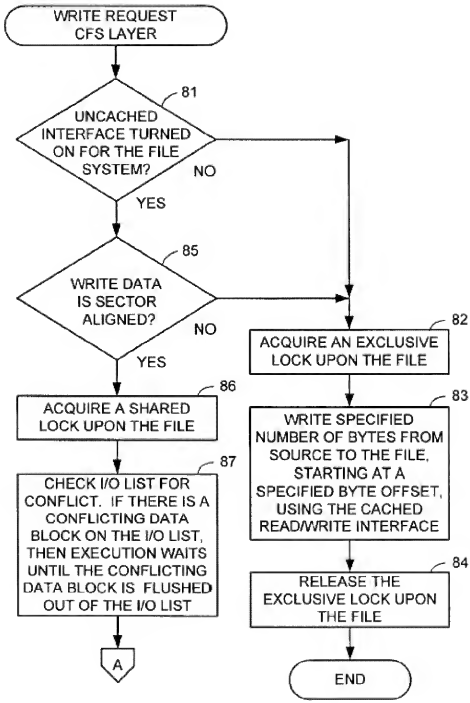
WRITE REQUEST CFS LAYER
81
UNCACHED INTERFACE TURNED ON FOR THE FILE SYSTEM?
NO
YES
85
WRITE DATA IS SECTOR ALIGNED?
NO
YES
82
ACQUIRE AN EXCLUSIVE LOCK UPON THE FILE
83
WRITE SPECIFIED NUMBER OF BYTES FROM SOURCE TO THE FILE, STARTING AT A SPECIFIED BYTE OFFSET, USING THE CACHED READ/WRITE INTERFACE
84
RELEASE THE EXCLUSIVE LOCK UPON THE FILE
END
86
ACQUIRE A SHARED LOCK UPON THE FILE
87
CHECK I/O LIST FOR CONFLICT. IF THERE IS A CONFLICTING DATA BLOCK ON THE I/O LIST, THEN EXECUTION WAITS UNTIL THE CONFLICTING DATA BLOCK IS FLUSHED OUT OF THE I/O LIST
A

FIG. 5

metadata block for the file (123 in FIG. 13; 141 in FIG. 16), the original indirect block of the file having been shared between the file and a read-only version of the file. (Applicants' specification, page 15 lines 8-11; page 22 line 22 to page 31 line 7; page 28 line 22 to page 29 line 7.)

The inventions of appellants' dependent claims 4 and 36 further include concurrent writing for more than one client to the metadata block for the file. (Applicants' specification, page 15, lines 17-22.)

The inventions of appellants' dependent claims 5 and 37 further include asynchronous writing to the file including a partial write to a new block (123 in FIG. 13) that has been copied at least in part from an original block (121 in FIG. 13) of the file, and wherein the method further includes checking a partial block conflict queue (73 in FIG. 4; page 14 lines 19-22; page 15 lines 17-22) for a conflict with a concurrent write to the new block (step 151 in FIG. 17; page 34 lines 7-10), and upon failing to find an indication of a conflict with a concurrent write to the new block, preallocating the new block (step 102 in FIG. 11), copying at least a portion of the original block of the file to the new block (step 153 in FIG. 17), and performing the partial write to the new block (step 154 in FIG. 17). (Appellants' specification, page 27 lines 4-6, page 28 line 22 to page 29 line 7, page 34 line 7 to page 35 line 5.)

The inventions of appellants' dependent claims 6 and 38 are similar to claim 13 to the extent that claims 6 and 38 further define that the asynchronous writing to the file includes a partial write to a new block (123 in FIG. 13) that has been copied at least in part from an original block (121 in FIG. 13) of the file, and wherein the method further includes checking a partial block conflict queue (73 in FIG. 4; page 14 lines 19-22; page 15 lines 17-22) for a conflict with a concurrent write to the new block (step 151 in FIG. 17; page 34 lines 7-10), and upon finding an indication of a conflict with a concurrent write to the new block, waiting until resolution of the conflict with the concurrent write to the new block (step 156 in FIG. 17; page 34 line 24 to page 35 line 2), and then performing the partial write to the new block (step 157 in FIG. 17; page 35 lines 2-5).

The inventions of appellants' dependent claims 11 and 45 are similar to claim 15 to the extent that claims 11 and 45 further define gathering together preallocated metadata blocks for a plurality of client write requests to the file (step 117 in FIG. 12), and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining the lock for the file (step 106 in FIG. 11), committing the gathered preallocated metadata blocks for the plurality of client write requests to the file (step 107 in FIG. 11; step 118 in FIG. 12), and then releasing the lock for the file (step 108 in FIG. 11). (Appellants' specification, page 4, lines 18-23; page 27 lines 14-23; page 28, lines 9-21.)

The inventions of appellants' dependent claims 12, 16 and 46 further include checking whether a previous commit is in progress (step 111 in FIG. 12) after asynchronously writing to the file (steps 103 and 104 in FIG. 11) and before obtaining the lock for the file for committing the metadata block to the file (step 112 or step 118 in FIG. 12), and upon finding that a previous commit is in progress, placing a request for committing the metadata block to the file on a staging queue (76 in FIG. 4) for the file (step 117 in FIG. 12). (Appellants' specification, page 28 lines 1-4 and 9-21.)

The inventions of appellants' dependent claim 17 is similar to claim 25 to the extent that claim 17 further defines the network file server responding to concurrent write requests by writing new data for specified blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache. (Steps 515 and 516 in FIG. 10; page 23 lines 19-23 and page 24 lines 7-15.)

The invention of appellants' dependent claim 18 is similar to claim 25 to the extent that claim 18 further defines the network file server responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing

to the file system cache a file block that has become stale. (Steps 92, 97, 513, 98 in FIG. 10; page 23 lines 5-17 page 24 lines 16-22.)

The inventions of appellant's dependent claims 19, 26 and 52 further include the network file server checking a read-in-progress flag for a file block (step 510 in FIG. 10) upon finding that the file block is not in the file system cache (step 92 in FIG. 10), and upon finding that the read-in-progress flag indicates that a prior read of the file block is in progress from the file system in the disk storage, waiting for completion of the prior read of the file block from the file system in the disk storage (step 511 in FIG. 10), and then again checking whether the file block is in the file system cache (step 92 in FIG. 10). (Applicants' specification, page 22 line 19 to page 23 line 9.)

The inventions of appellant's dependent claims 20, 27 and 53 further includes the network file server setting a read-in-progress flag for a file block (step 512 in FIG. 10) upon finding that the file block is not in the file system cache (step 92 in FIG. 10) and then beginning to read the file block from the file system in disk storage (step 512 in FIG. 10), clearing the read-in-progress flag upon writing to the file block on disk (steps 515 and 516 in FIG. 10), and inspecting the read-in-progress flag (step 513 in FIG. 10) to determine whether the file block has become stale due a concurrent write to the file block. (Applicants' specification, page 23 line 10 to page 24 line 6.)

The inventions of appellant's dependent claim 21, 28 and 54 further include the network file server maintaining a generation count (step 512 in FIG. 10) for each read of a file block from the file system in the disk storage in response to a read request for a file block that is not in the file system cache (step 92 in FIG. 10), and checking whether a file block having been read from the file system in the disk storage has become stale by checking whether the generation count for the file block having been read from the file system is the same as the generation count for the last read request for the same file block (step 513 in FIG. 10).

The invention of appellants' dependent claim 59 is similar to claim 61 to the extent that claim 59 further defines an uncached write interface (63 in FIG. 3), a file system cache (51 in FIG. 2 and FIG. 3) and a cached read-write interface (61 in FIG. 3; appellants' specification, page 7, lines 12-13; page 13 line 8 to page 14 line 17) wherein the uncached write interface bypasses the file system cache for sector-aligned write operations. (FIG. 3; step 85 in FIG. 5; page 13 lines 17-23; page 17 lines 12-15.)

The invention of appellants' dependent claim 60 is similar to claim 61 to the extent that claim 60 defines that the network file server is further programmed to invalidate cache blocks in the file system cache including sectors being written to by the uncached write interface. (FIG. 6, steps 88 and 89; appellants' specification, page 18 lines 8-14.)

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

1. Whether claims 1-4, 10, 11, 15, 22, 32, 33-36, 44, 45, 49, and 58-65, 67-71, and 73 are unpatentable under 35 U.S.C. 102(b) as being anticipated by Xu et al. U.S. Patent 6,324,581 B1.

2. Whether claims 5-9, 12-14, 16-21, 23-28, 37-43, 46-48, and 50-54 are unpatentable under 35 U.S.C. 103(a) over Xu et al. U.S. Patent 6,324,581 B1 in view of Marcotte U.S. Patent 6,449,614 B1.

VII. ARGUMENT

1. Claims 1-4, 10, 11, 15, 22, 32, 33-36, 44, 45, 49, and 58-65, 67-71, and 73 are patentable under 35 U.S.C. 102(b) and are not anticipated by Xu et al. U.S. Patent 6,324,581 B1.

"For a prior art reference to anticipate in terms of 35 U.S.C. § 102, every element of the claimed invention must be identically shown in a single reference." Diversitech Corp. v. Century Steps, Inc., 7 U.S.P.Q.2d 1315, 1317 (Fed. Cir. 1988), quoted in In re Bond, 910 F.2d 831,15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990) (vacating and remanding Board holding of anticipation; the elements must be arranged in the reference as in the claim under review, although this is not an *ipsis verbis* test).

Claim Interpretation Issue

With respect to appellants' claim 1, page 20 of the final Official Action of says: "Nothing within the claim limits the limitations to execute one after the other in a specific order." Appellants respectfully disagree. The word "then" appears at the end of each of steps (a), (b), (c), (d), (e), and (f). It is unreasonable to render the word "then" meaningless. It is also unreasonable to construe step (a) to be the same step as step (e), and it is unreasonable to construe step (c) to be the same step as step (g).

"While not an absolute rule, all claim terms are presumed to have meaning in a claim." Innova/Pure Water v. Safari Water Filtration Sys., Inc., 381 F.3d 1111, 1119 (Fed. Cir. 2004)(defendant's claim construction impermissibly read the term "operatively" out of the phrase "operatively connected"). Moreover, if the specification does not reveal any special definition for the phrase "; and then", then the phrase must be construed according to its ordinary meaning that the term would have to a person of ordinary skill in the art in question at the time of the invention. Phillips v. AWH Corp., 415 F.3d 1303, 1312-13 (Fed. Cir. 2005)(en banc). Dictionaries are among the many tools that can assist the court in determining the meaning of particular terminology to those of skill in the art of the invention. Id., at 1318.

Applicants respectfully submit that the ordinary meaning of the phrase "X, and then Y" is that Y is "following next after [X] in order". This ordinary meaning also avoids an issue of "double inclusion" that arises if step (a) were construed to be the same step as step (e), and step (c) were construed to be the same step as step (g).. Applicants respectfully submit that it is unreasonable to construe claim 1 so as to raise an issue of "double inclusion." See MPEP 2173.05(o) Double Inclusion. Appellants respectfully submit that similar language in their other claims should be construed in the same fashion.

Claims 1, 2, 10, 62, 68

Appellants' claim 1 calls for a network file server responding to a concurrent write request from a client for access to a file by a sequence of seven specific steps performed in a specific order. The network file server responds by:

(a) obtaining a lock for the file and then

(b) preallocating a metadata block for the file; and then

(c) releasing the lock for the file; and then

(d) asynchronously writing to the file; and then

(e) obtaining the lock for the file; and then

(f) committing the metadata block to the file; and then

(g) releasing the lock for the file.

The appellants' drawings show these steps (a) to (g) in FIG. 11 in boxes 101, 102, 103, 104-105, 1-6, 107, and 108, respectively, as described in appellants' specification on page 27 lines 3-23.

With respect to appellants' step (a) of obtaining a lock for the file, page 2 of the final Official action cites Xu column 3, line 60 through column 4, line 12. However, this same passage is cited on page 3 of the final Official Action for appellants' step (e) of obtaining a lock for the file. With respect to appellants' step (b) of preallocating a metadata block for the file, page 2 of the final Official Action cites Xu column 8, lines 36-64. With respect to appellants' step (c) of releasing the lock for the file, the final Official Action cites Xu column 9, lines 21-39. However, this releasing of a lock for the file in Xu occurs after writing data to the file, and not before as recited in appellants' claim 1 steps (c) and (d), and also this same passage is cited on page 3 of the final Official Action for appellants' step (g) of releasing the lock for the file. With respect to appellants' step (d) of asynchronously writing to the file, page 3 of the final Official Action again cites Xu column 9, lines 21-39. However, this writing occurs in Xu when the lock

of Xu column 3, line 60 through column 4, line 12 is held by the data mover that accesses the data storage locations specified by the metadata of the file. With respect to appellants' step (e) of obtaining the lock for the file, page 3 of the final Official Action again cites Xu column 3, line 60 though column 4, line 12, which occurs in Xu before the writing of Xu column 9, lines 21-29, instead of after, as recited in appellants' steps (d) and (e). With respect to appellants' step (f) of committing the metadata block to the file, page 3 of the final Official Action cites Xu column 8, lines 65 through column 9, line 39. With respect to appellants' step (g) of releasing the lock for the file, page 3 of the final Official Action again cites Xu column 9, lines 65 21-39.

It is respectfully submitted that Xu fails to disclose the appellants' step (c) of releasing the lock for the file before the step (d) of asynchronously writing to the file for Xu's network file server responding to a write request from a client. Xu also fails to disclose the appellants' step (e) of obtaining the lock for the file after the step (d) of asynchronously writing to the file. Nor are appellants' steps (c) and (e) inherent or obvious from Xu because Xu's lock of column 3, line 60 through column 4, line 12 is a lock granted by a first data mover computer to a second data mover so that the second data mover may access data storage locations of the file. The lock on the file as recited in appellants' claim 1 is not a lock for accessing data storage location of the file because it is released in appellants' step (c) prior to appellant's step (d) of asynchronously writing to the file, and again obtained in appellants' step (e) after appellant's step (d) of asynchronously writing to the file. Instead, the lock on the file as recited in appellants' claim 1 is associated with the step (b) of preallocating a metadata block for the file and is associated with the step (f) of committing the metadata block to the file, because the lock on the file as recited in

appellants' claim 1 is obtained in step (a) before step (b) and then released in step (c) after step (b), and it is again obtained in step (e) before step (f) and then released in step (g).

Nor does Xu need or suggest such a lock on the file to be associated with the step (b) of preallocating a metadata block for the file and associated with the step (f) of committing the metadata block to the file in this fashion as claimed in claim 1 because Xu discloses an entirely different mechanism to control access to preallocating metadata blocks for the file and committing metadata blocks to the file. This entirely different mechanism uses file system configuration by assigning a data mover to each file so that this Owner data mover has exclusive control over block allocation for the file and committing the metadata blocks to the file. See Xu col. 8, lines 47-54 and Xu col. 33 line 65 to col. 34 line 9. As disclosed in Xu col. 8 lines 36-54:

> In contrast to FIG. 1, the network file server architecture in FIG. 2 includes a data bypass path 48 between the first data mover 41 and the second file system 44 in order to bypass the second data mover 42, and a data bypass path 49 between the second data mover 42 and the first file system 43 in order to bypass the first data mover 41. It is possible for each of the data movers 41, 42 to access data in each of the file systems 43, 44, but if a data mover does not own the file access information for the file system to be accessed, then the data mover should ask the owner for permission to access the file system, or else a data consistency problem may arise. For example, when the first data mover 41 receives a file access request from its client 46, it accesses its directory of file ownership information to determine whether or not it owns the file system to be accessed. If the first data mover 41 does not own the file system to be accessed, then the first data mover 41 sends a metadata request to the data mover that owns the file system to be

> accessed. For example, if the first client 46 requests access to the second file system 44, then the first data mover 41 sends a metadata request to the second data mover 42.

As disclosed in Xu col. 33 line 56 to col. 34 line 9:

> There are several ways that the Owner can allocate disk blocks for the secondary data mover. In a preferred implementation, the secondary data mover tells the Owner that it wants to grow the file for some number of disk blocks. The Owner does the blocks allocation and allocates proper indirect blocks for the growth and informs the secondary data mover. Therefore, a secondary data mover works on the new file's metadata During this allocation, the blocks are not logged inside the log on the Owner and the file's in-memory inode structure is neither changed nor logged. When the secondary data mover sends the metadata back, the inode and indirect blocks are updated and logged. Some unused blocks are also reclaimed, because the free disk blocks are not shareable across different files inside one shadow file system. This makes ShFS's behavior different from that of UFS. Since SHFS does not have the usual file system structures, it does not support many of the normal file system operations, like name lookup. For those operations, ShFS can just return a proper error code as SFS currently does.

Claim 3

With respect to appellants' dependent claim 3, page 3 of the final Official Action cites col. 8 line 65 through col. 9 line 39 for a teaching of copying data from an original indirect block for the file to the metadata block for the file, the original indirect block of the file having been shared between the file and a read-only version of the file. However, it is not understood how

this passage or any other passage in Xu discloses "the original indirect block of the file having been shared between the file and a read-only version of the file." For example, compare Xu to appellants' FIG. 16 and the written description in appellants' specification on page 15 lines 8-11.

Pages 21 to 22 of the final Official Action responds to applicants' argument by saying: "Specifically, Xu discloses that during a write lock is obtained on the file system by disclosing the release of the lock in column 9 line 36. This suggests that the file system copy of the data is read-only to everyone other than the owner." However, the rejection of claim 3 over Xu is under 35 U.S.C. 102, not 35 U.S.C. 103. More importantly, applicants' claim 3 is defining that before the copying of the data, the file and the read-only version of the file shared the original indirect block of the file, and after the copying of the data, the copied data is in the metadata block of the file. Xu col. 8 line 65 through col. 9 line 39 discloses that the owner is reading the file metadata for others and writing file metadata for others so that the others are not reading or writing the file metadata.

Claim 4

With respect to dependent claim 4, page 3 of the final Official Action cites Xu col. 11 lines 20-37 and column 13 lines 36-42. However, these passages relate to shared data access among clients, and do not disclose details of write access to a metadata block for a file. As discussed above with respect to Xu col. 33 line 56 to col. 34 line 9, for each file, a particular data mover owner of the file is assigned by way of file system configuration to have exclusive control over write access to a metadata block for a file. If another client or data mover (called a

secondary data mover) is to perform a write operation on the file that would need to change the metadata block for a file, the secondary gets the metadata and a write lock on the file, but still the secondary sends the new metadata back to the Owner and the Owner updates and logs the inode and indirect block. It is not seen where Xu discloses that writing to such an inode or indirect block would occur concurrently for more than one client instead of serially by the Owner having exclusive access to the metadata blocks of the file.

Claim 11

With respect to appellants' dependent claim 11, pages 3-4 of the final Official Action cites Xu col. 8 line 36 through column 9, line 39. However, these passages fail to disclose gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining the lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file. Instead, as discussed above with respect to appellants' claim 4, as disclosed in Xu col. 33 line 36 to col. 34 line 9, for each file, a particular data mover owner of the file is assigned by way of file system configuration to have exclusive control over write access to a metadata block for a file. If another client or data mover (called a secondary data mover) is to perform a write operation on the file that would need to change the metadata block for a file, the secondary gets the metadata and a write lock on the file, but still the secondary sends the new metadata back to the Owner and the Owner updates and logs the inode and indirect block. It is not seen

where Xu discloses that such a metadata block update sequence would include gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining the lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file. In addition, if the Owner would gather together preallocated metadata blocks for a plurality of client write requests to the file, it would not need to obtain a lock on the file to commit the gathered preallocated metadata blocks because the Owner has been configured to have exclusive access to the metadata of the file.

In reply to the comments on pages 23-24 of the final Official Action, appellants respectfully submit that the read or write data cached by the data mover that does not own a file (Xu col. 9 lines 36) is not data of "preallocated metadata blocks for a plurality of client write requests." Instead, Xu col. 9 lines 21-26 discloses that "after the write data is committed to the second file system 44, the second data mover 42 commits any new file attributes by writing the new file attributes to the file system."

Claim 15, 22, 64, 70

With respect to appellants' independent claim 15, as discussed above with reference to appellants' claim 11, it is respectfully submitted that Xu fails to disclose gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the preallocated metadata blocks for the plurality of client write requests to the file by

obtaining the lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file.

Claims 32, 67, and 73

With respect to appellants' independent claim 32, as discussed above with reference to appellants' claim 1, it is respectfully submitted that Xu fails to disclose appellants' step c) of releasing the allocation mutex of the file [prior to the step d) of issuing asynchronous write requests for writing to the file], and appellants' step f) of obtaining the allocation mutex for the file [after the step d) of issuing asynchronous write requests for writing to the file]. In addition, the lock of Xu column 3, line 60 through col. 4, line 12 appears to be a read lock or a write lock, and not an allocation mutex. Moreover, as discussed above with reference to Xu col. 8, lines 47-54 and Xu col. 33 line 65 to col. 34 line 9, a data mover Owner of a file does not need an allocation mutex because it has been configured to have exclusive ownership and access to the metadata blocks of the file.

Claims 33, 34, and 44

Appellants respectfully submit that their independent apparatus claim 33 should be construed in the same fashion as their corresponding method claim 1 so that the claimed apparatus when operated performs the recited steps (a), (b), (c), (d), (e), (f), and (g) in the specified order. WMS Gaming., Inc., v. International Game Technology, 184 F.3d 1339, 1348 (Fed. Cir. 1999)("A general purpose computer, or microprocessor, programmed to carry out an

algorithm creates 'a new machine, because a general purpose computer in effect becomes a special purpose computer once it is programmed to perform particular functions pursuant to instructions from program software.'"(citations omitted)); In re Bernhart, 417 F.2d 1395, 1399-1400, 163 U.S.P.Q. 611, 615-16 (C.C.P.A. 1969). Therefore claims 33 and 34 are distinguished from Xu for the reasons given above with reference to claim 1.

Claim 35

Appellant's dependent claim 35 distinguishes Xu by virtue of the limitations incorporated by reference from its base claim 33 for the reasons given above with respect to claim 33. In addition, claim 35 includes limitations similar to claim 3 and therefore further distinguishes Xu for the reasons give above with respect to claim 3.

Claim 36

Appellant's dependent claim 36 distinguishes Xu by virtue of the limitations incorporated by reference from its base claim 33 for the reasons given above with respect to claim 33. In addition, claim 36 includes limitations similar to claim 4 and therefore further distinguishes Xu for the reasons give above with respect to claim 4.

Claim 45

Appellant's dependent claim 45 distinguishes Xu by virtue of the limitations incorporated by reference from its base claim 33 for the reasons given above with respect to claim 33. In addition, claim 36 includes limitations similar to claim 11 and therefore further distinguishes Xu for the reasons give above with respect to claim 11.

Claim 49

Appellants respectfully submit that their independent apparatus claim 49 should be construed in the same fashion as their method claim 15 so that the claimed apparatus when operated performs the recited steps (a), (b), (c), (d), (e), (f), and (g) in the specified order. WMS Gaming, supra. Therefore claim 49 is distinguished from Xu for the reasons given above with reference to claim 1.

Claims 58, 59

Appellants respectfully submit that their independent apparatus claim 58 should be construed in the same fashion as their method claim 32 so that the claimed apparatus when operated performs the recited steps (a), (b), (c), (d), (e), (f), (g), (h) in the specified order. WMS Gaming, supra. Therefore claim 58 is distinguished from Xu for the reasons given above with reference to claim 32.

Claim 60

Appellants' dependent claim 60 distinguishes Xu by virtue of its limitations incorporated from its base claim 58 for the reasons discussed above with respect to claim 58. Claim 60 further distinguishes Xu by reciting that the network file server is further programmed to invalidate cache blocks in the file system cache including sectors being written to by the uncached write interface. It is not seen where Xu discloses that a network file server that is programmed to invalidate cache blocks in the file system cache including sectors being written to by an uncached write interface. The description in Xu col. 9 lines 30-39 relates to operation of a cached interface.

Claim 61

Claim 61 distinguishes Xu by reciting that the network file server is further programmed to invalidate cache blocks in the file system cache including sectors being written to by the uncached write interface. It is not seen where Xu discloses that a network file server that is programmed to invalidate cache blocks in the file system cache including sectors being written to by an uncached write interface. The description in Xu col. 9 lines 30-39 relates to operation of a cached interface.

Claims 63 and 69

Appellants' dependent claims 63 and 69 are dependent upon appellants' claim 13, and therefore are distinguished from Xu because Xu fails to explicitly recite a number of limitations incorporated by reference from claim 13, such as "the asynchronous writing to the file including a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method further includes checking a partial block conflict queue for a conflict with a concurrent write to the new block, and upon finding an indication of a conflict with a concurrent write to the new block, waiting until resolution of the conflict with the concurrent write to the new block, and then performing the partial write to the new block" as noted in the first paragraph on page 12 of the final Official Action. Appellants respectfully submit that claims 63 and 69 are patentable over the proposed combination of Xu and Marcotte for the reasons discussed below with respect to their base claim 13.

Claims 65 and 71

Appellants' dependent claims 65 and 71 are dependent upon appellants' claim 25, and therefore are distinguished from Xu because Xu fails to explicitly recite a number of limitations incorporated by reference from claim 25, such as "the network file server responding to concurrent write requests by writing new data for specified blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache," as noted in the middle of page 17 of the final Official Action. Appellants respectfully submit that claims 65 and 71 are

patentable over the proposed combination of Xu and Marcotte for the reasons discussed below with respect to their base claim 25.

2. Claims 5-9, 12-14, 16-21, 23-28, 37-43, 46-48, and 50-54 are patentable under 35 U.S.C. 103(a) over Xu et al. U.S. Patent 6,324,581 B1 in view of Marcotte U.S. Patent 6,449,614 B1.

The policy of the Patent and Trademark Office has been to follow in each and every case the standard of patentability enunciated by the Supreme Court in Graham v. John Deere Co., 148 U.S.P.Q. 459 (1966). M.P.E.P. § 2141. As stated by the Supreme Court:

> Under § 103, the scope and content of the prior art are to be determined; differences between the prior art and the claims at issue are to be ascertained; and the level of ordinary skill in the pertinent art resolved. Against this background, the obviousness or nonobviousness of the subject matter is determined. Such secondary considerations as commercial success, long felt but unsolved needs, failure of others, etc., might be utilized to give light to the circumstances surrounding the origin of the subject matter sought to be patented. As indicia of obviousness or nonobviousness, these inquiries may have relevancy.

148 U.S.P.Q. at 467.

The problem that the inventor is trying to solve must be considered in determining whether or not the invention would have been obvious. The invention as a whole embraces the structure, properties and problems it solves. In re Wright, 848 F.2d 1216, 1219, 6 U.S.P.Q.2d 1959, 1961 (Fed. Cir. 1988).

Marcotte discloses an interface system and methods for asynchronously updating a share resource with locking facility. Tasks make updates requested by calling tasks to a shared resource serially in a first come first served manner, atomically, but not necessarily synchronously, such that a current task holding an exclusive lock on the shared resource makes the updates on behalf of one or more calling tasks queued on the lock. Updates waiting in a queue on the lock to the shared resource may be made while the lock is held, and others deferred for post processing after the lock is released. Some update requests may also, at the calling application's option, be executed synchronously. Provision is made for nested asynchronous locking. Data structures (wait_elements) describing update requests may queued in a wait queue for update requests awaiting execution by a current task, other than the calling task, currently holding an exclusive lock on the shared resource. Other queues are provided for queuing data structures removed from the wait queue but not yet processed; data structures for requests to unlock or downgrade a lock; data structures for requests which have been processed and need to be returned to free storage; and data structures for requests that need to be awakened or that describe post processing routines that are to be run while the lock is not held. (Abstract.)

Claims 8, and 9

Appellants' dependent claims 8, and 9 depend on claim 1. Xu has been distinguished above with respect to the base claim 1, and Marcotte does not provide the limitations of these independent claims that are missing from Xu. Therefore the dependent claims 8 and 9 are patentable over the proposed combination of Xu and Marcotte. It is respectfully submitted that it

would not have been obvious for one of ordinary skill to combine Xu and Marcotte in the fashion as proposed in the final Official Action and then modify that combination by adding the missing limitations.

Claims 23 and 24

Appellants' dependent claims 23 and 24 depend on independent claim 15. Xu has been distinguished above with respect to the base claim 15, and Marcotte does not provide the limitations of these independent claims that are missing from Xu. Therefore the dependent claims 23 and 24 are patentable over the proposed combination of Xu and Marcotte. It is respectfully submitted that it would not have been obvious for one of ordinary skill to combine Xu and Marcotte in the fashion as proposed in the final Official Action and then modify that combination by adding the missing limitations.

Claims 40, 41, 42, 43

Appellants' dependent claims 40, 41, 42, 43 depend directly or indirectly on independent claim 33. Xu has been distinguished above with respect to the base claim 33, and Marcotte does not provide the limitations of these independent claims that are missing from Xu. Therefore the dependent claims 23 and 24 are patentable over the proposed combination of Xu and Marcotte. It is respectfully submitted that it would not have been obvious for one of ordinary skill to combine Xu and Marcotte in the fashion as proposed in the final Official Action and then modify that combination by adding the missing limitations.

Claim 5

Xu has been distinguished above with respect to the independent base claim 1 and Marcotte does not provide the limitations of claim 1 that is missing from Xu.

Appellants' dependent claim 5 adds to claim 1 the limitations of "wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method further includes checking a partial block conflict queue for a conflict with a concurrent write to the new block, and upon failing to find an indication of a conflict with a concurrent write to the new block, preallocating the new block, copying at least a portion of the original block of the file to the new block, and performing the partial write to the new block." Appellants' partial block conflict queue 73 is shown in appellants' FIG. 4 and described in appellant's specification on page 15 lines 17-22 as follows:

> The preallocation method allows concurrent writes to indirect blocks within the same file. Multiple writers can write to the same indirect block tree concurrently without improper replication of the indirect blocks. Two different indirect blocks will not be allocated for replicating the same indirect block. The write threads use the partial block conflict queue 73 and the partial write wait queue 74 to avoid conflict during partial block write operations, as further described below with reference to FIG. 13.

See also appellants' FIG. 13 and specification page 28 line 22 to page 29 line 7; and FIG. 17 and page 34 line 7 to page 35 line 5.

With respect to appellants' dependent claim 5, page 8 of the final Official Action recognizes that Xu fails to explicitly recite the limitations added by dependent claim 5, and says that Marcotte teaches these limitations, citing column 13, lines 35 through col. 14, line 43. However, Marcotte column 13, lines 35 through col. 14, line 43 deals with managing an I/O device holding queue above a device for queuing pending I/O's if the number of I/O's issued to the device exceeds an a threshold that is adjustable by an application. It is not seen where Marcotte discloses a partial write to a new block that has been copied at least in part from an original block of the file. Nor is it seen where Marcotte discloses checking a partial block conflict queue for a conflict with a concurrent write to the new block, and upon failing to find an indication of a conflict with a concurrent write to the new block, preallocating the new block, copying at least a portion of the original block of the file to the new block, and performing the partial write to the new block.

"[R]ejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness." In re Kahn, 441 F. 3d 977, 988 (Fed. Cir. 2006). A fact finder should be aware of the distortion caused by hindsight bias and must be cautious of arguments reliant upon ex post reasoning. See KSR International Co. v. Teleflex Inc., 550 U.S. __, 82 USPQ2d 1385 (2007)), citing Graham, 383 U. S. at 36 (warning against a "temptation to read

into the prior art the teachings of the invention in issue" and instructing courts to "guard against slipping into the use of hindsight.").

Claims 6 and 7

.Xu has been distinguished above with respect to the independent base claim 1 and Marcotte does not provide the limitations of claim 1 that are missing from Xu.

Appellants' claim 6 is similar to claim 5 in that it also adds to claim 1 the limitations of wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method further includes checking a partial block conflict queue for a conflict with a concurrent write to the new block. With respect to dependent claim 6, page 9 of the Official Action also recognizes that Xu fails to explicitly recite the limitations added by dependent claim 6, and says that Marcotte teaches these limitations, again citing column 13, lines 35 through col. 14, line 43. However, Marcotte column 13, lines 35 through col. 14, line 43 deals with managing an I/O device holding queue above a device for queuing pending I/O's if the number of I/O's issued to the device exceeds an a threshold that is adjustable by an application. It is not seen where Marcotte discloses a partial write to a new block that has been copied at least in part from an original block of the file. Nor is it seen where Marcotte discloses checking a partial block conflict queue for a conflict with a concurrent write to the new block.

Claim 7 is dependent upon claim 6 and therefore distinguishes Xu for the same reasons as claim 6.

Claim 12

Xu has been distinguished above with respect to the independent base claim 1 and Marcotte does not provide the limitations of claim 1 that are missing from Xu.

With respect to the limitations added in claim 12, appellants respectfully submit that Xu does not further teach checking whether a previous commit is in progress after asynchronously writing to the file and before obtaining the lock for the file for committing the metadata block to the file. As discussed above with reference to appellants' claim 1, in response to a concurrent write request from a client, Xu does not obtain a lock for the file after the asynchronously writing to the file. Nor does Marcotte column 12, line 7 through column 13, line 32 disclose these limitations missing from Xu or specifically deal with committing "metadata blocks" to a file.

Claim 13 and 14

Appellants' independent claim 13 recites "wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method includes checking a partial block conflict queue for a conflict with a concurrent write to the new block, and upon finding an indication of a conflict with a concurrent write to the new block, waiting until resolution of the conflict with the concurrent write to the new block, and then performing the partial write to the new block." Thus, Appellants' independent claim 13 is patentable over Xu in combination with Marcotte for the reasons given above with reference to appellants' claim 5.

Claim 16

Appellants' dependent claim 16 adds to claim 15 the limitations of claim 12 and therefore is patentable over Xu in combination with Marcotte for the reasons given above with reference to appellants' claims 12 and 15.

Claim 17.

Appellants' dependent claim 17 adds to claim 15 the limitations of "wherein the network file server includes disk storage containing a file system, and a file system cache storing data of blocks of the file, and the method further includes the network file server responding to concurrent write requests by writing new data for specified blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache." Page 14 of the Official Action recognizes that Xu fails to explicitly recite these limitations, and cites Marcotte column 12 line 7 through column 13, line 32. However, it is not understood how the appellants' specific claim limitations added by claim 17 are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Claim 18

Appellants' dependent claim 18 adds to claim 17 the limitations of "the network file server responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks

have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing to the file system cache a file block that has become stale." Page 15 of the Official Action again cites Marcotte column 12 line 7 through column 13, line 32. However, it is not understood how the appellants' specific claim limitations added in claim 18 are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Claim 19

Appellants' dependent claim 19 adds to claim 18 the limitations of "the network file server checking a read-in-progress flag for a file block upon finding that the file block is not in the file system cache, and upon finding that the read-in-progress flag indicates that a prior read of the file block is in progress from the file system in the disk storage, waiting for completion of the prior read of the file block from the file system in the disk storage, and then again checking whether the file block is in the file system cache." Page 15 of the Official Action again cites Marcotte column 12 line 7 through column 13, line 32. However, it is not understood how the appellants' specific claim limitations added in claim 19 are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Claim 20

Appellants' dependent claim 20 adds to claim 18 the limitations of "the network file server setting a read-in-progress flag for a file block upon finding that the file block is not in the file system cache and then beginning to read the file block from the file system in disk storage,

clearing the read-in-progress flag upon writing to the file block on disk, and inspecting the read-in-progress flag to determine whether the file block has become stale due a concurrent write to the file block." Page 15 of the Official Action again cites Marcotte column 12 line 7 through column 12, line 21. However, it is not understood how the appellants' specific claim limitations added in claim 20 are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Claim 21

Appellants' dependent claim 21 adds to claim 18 the limitations of "the network file server maintaining a generation count for each read of a file block from the file system in the disk storage in response to a read request for a file block that is not in the file system cache, and checking whether a file block having been read from the file system in the disk storage has become stale by checking whether the generation count for the file block having been read from the file system is the same as the generation count for the last read request for the same file block." Page 16 of the Official Action again cites Marcotte column 12 line 7 through column 12, line 21. However, it is not understood how the appellants' specific claim limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Claim 25

With reference to appellants' independent claim 25, pages 17-18 of the final Official Action recognize that Xu fails to explicitly recite the network file server responding to concurrent write requests by writing new data for specified blocks of the file to disk storage

without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache. (See, e.g., appellants' FIG. 10, steps 515 and 516; appellants' spec., page 23 lines 19-23 and page 24 lines 7-15.) Page 17 of the final Official Action further recognizes that Xu fails to explicitly recite the network file server responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become state, and not writing to the file system cache a file block that has become stale. (See, e.g., appellants' FIG. 10, steps 92, 97, 513, 98; appellants' spec., page 23 lines 5-17; page 24 lines 16-22 .) Pages 18-19 of the final Official Action cites col. 12 line 7 through column 12, line 32 for all of these limitations not explicitly recited in Xu. However, it is not understood how all of the appellants' specific claim limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Marcotte column 12 line 7 through column 13, line 32, deals generally with maintaining a list of lock waiters. As shown in Marcotte FIG. 9, when a task waits for a lock, it queues its WAIT_ELEMENT to the lock using lock or wait routine 175 and also adds it WAIT_ELEMENT to a global list or queue 230 before it waits, and remotes it from the global list 230 after it waits. As shown in Marcotte FIG. 11, do_wait 240 is executed each time a thread needs to go into a wait for a lock. Step 241 executes a lock_UpdateResource procedure. Step 242 waits on ecb; and upon receiving it, step 243 executes the lock_Update Resource procedure. Marcotte says that in this way, the task waiting on a lock 100 will only actually suspend itself on the WAIT

call. Adding and removing tasks (WAIT ELEMENTS) from global list 230 is done in a manner guaranteed to make the calling task wait. FIG. 12 shows Referring to FIG. 12, add_to_global routine 245 (with waitp-arg) includes step 246 which determines if prevent flag is null; if so, step 248 posts an error code in the ecb field 108 of the WAIT_ELEMENT being processed; and, if not, step 247 adds the WAIT_ELEMENT (in this case, task 232) to the head of global list 230. FIG. 13 shows a remove_from_global routine 250 (with waitp=arg) including step 251 which determines if prevent flag 124 is null. If so, return code (rc) is set to zero; and if not, this WAIT_ELEMENT (say, 233) is removed from global list 230. In step 254, the return code (rc) is returned to the caller. FIG. 14 shows a return_wait routine 260 (with waitp=prc) including step 261 which determines if waitp is null. If not, the WAIT_ELEMENT pointed to by waitp is returned to free storage. Return wait 260 is the post processing routine for remove_from_global 250, and remove_from_global communicates the address of the WAIT_ELEMENT via its return code, that is input to return_wait (automatically by the resource update facility) as prc. Return_wait 260 returns the WAIT_ELEMENT to free storage. Since routine 260 is a post processing routine, the free storage return is NOT performed while holding the lock. This shows the benefits of a post processing routine, and passing the return value from a resource update routine to the post processing routine. FIG. 15 shows quiesce_global 265 wakes up all waiters and in step 267 tells them that the program is terminating due to error by way of prevent flag 124 being set to 1 in step 266. In step 268 pointer 231 and 234 are cleared so global list 230 is empty.

Thus, it is not understood how Marcotte's general teaching of a way of maintaining a list of lock waiters discloses the appellants' specific limitations of the network file server responding

to concurrent write requests by writing new data for specified blocks of the file to disk storage without writing the new data for the specified blocks of the file to the file system cache. invalidating the specified blocks of the file in the file system cache, and responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become state, and not writing to the file system cache a file block that has become stale.

"[R]ejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness." In re Kahn, 441 F. 3d 977, 988 (Fed. Cir. 2006). A fact finder should be aware of the distortion caused by hindsight bias and must be cautious of arguments reliant upon ex post reasoning. See KSR International Co. v. Teleflex Inc., 550 U.S. __, 82 USPQ2d 1385 (2007)), citing Graham, 383 U. S. at 36 (warning against a "temptation to read into the prior art the teachings of the invention in issue" and instructing courts to "guard against slipping into the use of hindsight.").

Claim 26

Appellant's dependent claim 26 adds to claim 25 limitations Appellants' dependent claim 19 adds to claim 18 the limitations of "the network file server checking a read-in-progress flag for a file block upon finding that the file block is not in the file system cache, and upon finding that the read-in-progress flag indicates that a prior read of the file block is in progress from the

file system in the disk storage, waiting for completion of the prior read of the file block from the file system in the disk storage, and then again checking whether the file block is in the file system cache." Page 18 of the Official Action cites Marcotte column 12 line 7 through column 13, line 32. However, it is not understood how these specific limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Claim 27

Appellants' dependent claim 27 adds to claim 25 the limitations of "the network file server setting a read-in-progress flag for a file block upon finding that the file block is not in the file system cache and then beginning to read the file block from the file system in disk storage, clearing the read-in-progress flag upon writing to the file block on disk, and inspecting the read-in-progress flag to determine whether the file block has become stale due a concurrent write to the file block." Page 19 of the final Official Action again cites Marcotte column 12 line 7 through column 13, line 32. However, it is not understood how these specific limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Claim 28

Appellants' dependent claim 28 adds to claim 25 the limitations of "the network file server maintaining a generation count for each read of a file block from the file system in the disk storage in response to a read request for a file block that is not in the file system cache, and checking whether a file block having been read from the file system in the disk storage has

become stale by checking whether the generation count for the file block having been read from the file system is the same as the generation count for the last read request for the same file block." Page 19 of the Official Action again cites Marcotte column 12 line 7 through column 12, line 21. However, it is not understood how these specific limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Claim 37

Appellants' dependent claim 37 is an apparatus claim corresponding to appellants' method claim 5. Therefore appellants respectfully submit that claim 37 is patentable over the proposed combination of Xu in view of Marcotte for the reasons given above with reference to claim 5.

Claims 38, 39

Appellants' dependent claim 38 is an apparatus claim corresponding to appellants' method claim 6. Therefore appellants respectfully submit that claim 37 is patentable over the proposed combination of Xu in view of Marcotte for the reasons given above with reference to claim 13. Claim 39 is dependent upon claim 39, and therefore is patentable over the proposed combination of Xu in view of Marcotte for the same reasons as claim 38.

Claim 46

Appellants' dependent claim 46 is an apparatus claim corresponding to appellants' method claim 12. Therefore appellants respectfully submit that claim 46 is patentable over the proposed combination of Xu in view of Marcotte for the reasons given above with reference to claim 12.

Claims 47 and 48

Appellants' independent claim 47 is an apparatus claim corresponding to appellants' independent method claim 13. Therefore appellants respectfully submit that claim 47 is patentable over the proposed combination of Xu in view of Marcotte for the reasons given above with reference to claim 13.

Claim 50

Appellants' dependent claim 50 is an apparatus claim dependent upon claim 49 and adds limitations corresponding to the limitations added in applicants' method claim 16. Therefore appellants respectfully submit that claim 46 is further patentable over the proposed combination of Xu in view of Marcotte for the reasons given above with reference to claim 16.

Claims 51

Appellants' independent claim 51 is an apparatus claim corresponding to appellants' independent method claim 25. Therefore appellants respectfully submit that claim 51 is

patentable over the proposed combination of Xu in view of Marcotte for the reasons given above with reference to claim 25.

Claim 52

Appellants' dependent claim 52 is an apparatus claim corresponding to applicants' method claim 26. Therefore appellants respectfully submit that claim 52 is further patentable over the proposed combination of Xu in view of Marcotte for the reasons given above with reference to claim 26.

Claim 53

Appellants' dependent claim 53 is an apparatus claim corresponding to applicants' method claim 27. Therefore appellants respectfully submit that claim 53 is further patentable over the proposed combination of Xu in view of Marcotte for the reasons given above with reference to claim 27.

Claim 54

Appellants' dependent claim 54 is an apparatus claim corresponding to applicants' method claim 28. Therefore appellants respectfully submit that claim 53 is further patentable over the proposed combination of Xu in view of Marcotte for the reasons given above with reference to claim 28.

In view of the above, the rejection of the claims should be reversed.

Respectfully submitted,

/ Richard C. Auchterlonie /

Richard C. Auchterlonie
Reg. No. 30,607
NOVAK DRUCE & QUIGG, LLP
1000 Louisiana, 53rd Floor
Houston, TX 77002
713-571-3460 (Telephone)
713-456-2836 (Telefax)
Richard.Auchterlonie@novakdruce.com

VIII. CLAIMS APPENDIX

The claims involved in this appeal are as follows:

1. A method of operating a network file server for providing clients with concurrent write access to a file, the method comprising the network file server responding to a concurrent write request from a client by:

(a) obtaining a lock for the file; and then

(b) preallocating a metadata block for the file; and then

(c) releasing the lock for the file; and then

(d) asynchronously writing to the file; and then

(e) obtaining the lock for the file; and then

(f) committing the metadata block to the file; and then

(g) releasing the lock for the file.

2. The method as claimed in claim 1, wherein the file further includes a hierarchy of blocks including an inode block of metadata, data blocks of file data, and indirect blocks of metadata, and wherein the metadata block for the file is an indirect block of metadata.

3. The method as claimed in claim 2, which further includes copying data from an original indirect block of the file to the metadata block for the file, the original indirect block of the file having been shared between the file and a read-only version of the file.

4. The method as claimed in claim 1, which further includes concurrent writing for more than one client to the metadata block for the file.

5. The method as claimed in claim 1, wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method further includes checking a partial block conflict queue for a conflict with a concurrent write to the new block, and upon failing to find an indication of a conflict with a concurrent write to the new block, preallocating the new block, copying at least a portion of the original block of the file to the new block, and performing the partial write to the new block.

6. The method as claimed in claim 1, wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method further includes checking a partial block conflict queue for a conflict with a concurrent write to the new block, and upon finding an indication of a conflict with a concurrent write to the new block, waiting until resolution of the conflict with the concurrent write to the new block, and then performing the partial write to the new block.

7. The method as claimed in claim 6, which further includes placing a request for the partial write in a partial write wait queue upon finding an indication of a conflict with a concurrent write to the new block, and performing the partial write upon servicing the partial write wait queue.

8. The method as claimed in claim 1, which further includes checking an input-output list for a conflicting prior concurrent access to the file, and upon finding a conflicting prior concurrent access to the file, suspending the asynchronous writing to the file until the conflicting prior concurrent access to the file is no longer conflicting.

9. The method as claimed in claim 8, which further includes providing a sector-level granularity of byte range locking for concurrent write access to the file by the suspending of the asynchronous writing to the file until the conflicting prior concurrent access is no longer conflicting.

10. The method as claimed in claim 1, which further includes writing the metadata block to a log in storage of the network file server for committing the metadata block for the file.

11. The method as claimed in claim 1, which further includes gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining the

lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file.

12. The method as claimed in claim 1, which further includes checking whether a previous commit is in progress after asynchronously writing to the file and before obtaining the lock for the file for committing the metadata block to the file, and upon finding that a previous commit is in progress, placing a request for committing the metadata block to the file on a staging queue for the file.

13. A method of operating a network file server for providing clients with concurrent write access to a file, the method comprising the network file server responding to a concurrent write request from a client by:

(a) preallocating a block for the file; and then

(b) asynchronously writing to the file; and then

(c) committing the block to the file;

wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method includes checking a partial block conflict queue for a conflict with a concurrent write to the new block, and upon finding an indication of a conflict with a concurrent write to the new block, waiting until resolution of the conflict with the concurrent write to the new block, and then performing the partial write to the new block.

14. The method as claimed in claim 13, wherein the method further includes placing a request for the partial write in a partial write wait queue upon finding an indication of a conflict with a concurrent write to the new block, and performing the partial write upon servicing the partial write wait queue.

15. A method of operating a network file server for providing clients with concurrent write access to a file, the method comprising the network file server responding to a concurrent write request from a client by:

(a) preallocating a metadata block for the file; and then

(b) asynchronously writing to the file; and then

(c) committing the metadata block to the file;

wherein the method includes gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining a lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file.

16. The method as claimed in claim 15, which further includes checking whether a previous commit is in progress after asynchronously writing to the file and before obtaining the lock for the file for committing the block to the file, and upon finding that a previous commit is in

progress, placing a request for committing the metadata block to the file on a staging queue for the file.

17. The method as claimed in claim 15, wherein the network file server includes disk storage containing a file system, and a file system cache storing data of blocks of the file, and the method further includes the network file server responding to concurrent write requests by writing new data for specified blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache.

18. The method as claimed in claim 17, which further includes the network file server responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing to the file system cache a file block that has become stale.

19. The method as claimed in claim 18, which further includes the network file server checking a read-in-progress flag for a file block upon finding that the file block is not in the file system cache, and upon finding that the read-in-progress flag indicates that a prior read of the file block is in progress from the file system in the disk storage, waiting for completion of the

prior read of the file block from the file system in the disk storage, and then again checking whether the file block is in the file system cache.

20. The method as claimed in claim 18, which further includes the network file server setting a read-in-progress flag for a file block upon finding that the file block is not in the file system cache and then beginning to read the file block from the file system in disk storage, clearing the read-in-progress flag upon writing to the file block on disk, and inspecting the read-in-progress flag to determine whether the file block has become stale due a concurrent write to the file block.

21. The method as claimed in claim 18, which further includes the network file server maintaining a generation count for each read of a file block from the file system in the disk storage in response to a read request for a file block that is not in the file system cache, and checking whether a file block having been read from the file system in the disk storage has become stale by checking whether the generation count for the file block having been read from the file system is the same as the generation count for the last read request for the same file block.

22. The method as claimed in claim 15, which further includes processing multiple concurrent read and write requests by pipelining the requests through a first processor and a second processor, the first processor performing metadata management for the multiple

concurrent read and write requests, and the second processor performing asynchronous reads and writes for the multiple concurrent read and write requests.

23. The method as claimed in claim 15, which further includes serializing the reads by delaying access for each read to a block that is being written to by a prior, in-progress write until completion of the write to the block that is being written to by the prior, in-progress write.

24. The method as claimed in claim 15, which further includes serializing the writes by delaying access for each write to a block that is being accessed by a prior, in-progress read or write until completion of the read or write to the block that is being accessed by the prior, in-progress read or write.

25. A method of operating a network file server for providing clients with concurrent read and write access to a file, the method comprising the network file server responding to a concurrent write request from a client by:

(a) preallocating a metadata block for the file; and then

(b) asynchronously writing to the file; and then

(c) committing the metadata block to the file;

wherein the network file server includes disk storage containing a file system, and a file system cache storing data of blocks of the file, and the method includes the network file server responding to concurrent write requests by writing new data for specified blocks of the file to the

disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache, and

which includes the network file server responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing to the file system cache a file block that has become stale.

26. The method as claimed in claim 25, which further includes the network file server checking a read-in-progress flag for a file block upon finding that the file block is not in the file system cache, and upon finding that the read-in-progress flag indicates that a prior read of the file block is in progress from the file system in the disk storage, waiting for completion of the prior read of the file block, and then again checking whether the file block is in the file system cache.

27. The method as claimed in claim 25, which further includes the network file server setting a read-in-progress flag for a file block upon finding that the file block is not in the file system cache and then beginning to read the file block from the file system in disk storage, clearing the read-in-progress flag upon writing to the file block on disk, and inspecting the read-in-progress flag to determine whether the file block has become stale due a concurrent write to the file block.

28. The method as claimed in claim 25, which further includes the network file server maintaining a generation count for each read of a file block from the file system in the disk storage in response to a read request for a file block that is not in the file system cache, and checking whether a file block having been read from the file system in the disk storage has become stale by checking whether the generation count for the file block having been read from the file system is the same as the generation count for the last read request for the same file block.

32. A method of operating a network file server for providing clients with concurrent write access to a file, the method comprising the network file server responding to a concurrent write request from a client by executing a write thread, execution of the write thread including:

(a) obtaining an allocation mutex for the file; and then

(b) preallocating new metadata blocks that need to be allocated for writing to the file; and then

(c) releasing the allocation mutex for the file; and then

(d) issuing asynchronous write requests for writing to the file;

(e) waiting for callbacks indicating completion of the asynchronous write requests; and then

(f) obtaining the allocation mutex for the file; and then

(g) committing the preallocated metadata blocks; and then

(h) releasing the allocation mutex for the file.

33. A network file server comprising storage for storing a file, and at least one processor coupled to the storage for providing clients with concurrent write access to the file, wherein the network file server is programmed for responding to a concurrent write request from a client by:

(a) obtaining a lock for the file; and then

(b) preallocating a metadata block for the file; and then

(c) releasing the lock for the file; and then

(d) asynchronously writing to the file; and then

(e) obtaining the lock for the file; and then

(f) committing the metadata block to the file; and then

(g) releasing the lock for the file.

34. The network file server as claimed in claim 33, wherein the file further includes a hierarchy of blocks including an inode block of metadata, data blocks of file data, and indirect blocks of metadata, and wherein the metadata block for the file is an indirect block of metadata.

35. The network file server as claimed in claim 34, which is further programmed for copying data from an original indirect block of the file to the metadata block for the file, the original indirect block of the file having been shared between the file and a read-only version of the file.

36. The network file server as claimed in claim 33, which is further programmed for concurrent writing for more than one client to the metadata block for the file.

37. The network file server as claimed in claim 33, which further includes a partial block conflict queue for indicating a concurrent write to a new block that is being copied at least in part from an original block of the file, and wherein the network file server is further programmed to respond to a client request for a partial write to the new block by checking the partial block conflict queue for a conflict, and upon failing to find an indication of a conflict, preallocating the new block, copying at least a portion of the original block of the file to the new block, and performing a partial write to the new block.

38. The network file server as claimed in claim 33, which further includes a partial block conflict queue for indicating a concurrent write to a new block that is being copied at least in part from an original block of the file, and wherein the network file server is further programmed to respond to a client request for a partial write to the new block by checking the partial block conflict queue for a conflict, and upon finding an indication of a conflict, waiting until resolution of the conflict with the concurrent write to the new block, and then performing the partial write to the new block.

39. The network file server as claimed in claim 38, which further includes a partial write wait queue, and wherein the network file server is further programmed for placing a request for the partial write in the partial write wait queue upon finding an indication of a conflict, and performing the partial write upon servicing the partial write wait queue.

40. The network file server as claimed in claim 33, which is further programmed for maintaining an input-output list of concurrent reads and writes to the file, and when writing to the file, for checking the input-output list for a conflicting prior concurrent read or write access to the file, and upon finding a conflicting prior concurrent read or write access to the file, suspending the asynchronous writing to the file until the conflicting prior concurrent read or write access to the file is no longer conflicting.

41. The network file server as claimed in claim 40, which is further programmed so that the suspending of the asynchronous writing to the file until the conflicting prior concurrent read or write access to the file is no longer conflicting provides a sector-level granularity of byte range locking for concurrent write access to the file.

42. The network file server as claimed in claim 33, which is further programmed for maintaining an input-output list of concurrent reads and writes to the file, and when reading from the file, for checking the input-output list for a conflicting prior concurrent write access to the file, and upon finding a conflicting prior concurrent write access to the file, suspending the reading to the file until the conflicting prior concurrent write access to the file is no longer conflicting.

43. The network file server as claimed in claim 42, which is further programmed so that the suspending of the reading to the file until the conflicting prior concurrent write access to the file

is no longer conflicting provides a sector-level granularity of byte range locking for concurrent read access to the file.

44. The network file server as claimed in claim 33, which is further programmed for committing the metadata block for the file by writing the metadata block to a log in the storage.

45. The network file server as claimed in claim 33, which is further programmed for gathering together preallocated metadata blocks for a plurality of client requests for write access to the file, and committing together the preallocated metadata blocks for the plurality of client requests for access to the file by obtaining the lock for the file, committing the gathered preallocated metadata blocks for the plurality of client requests for write access to the file, and then releasing the lock for the file.

46. The network file server as claimed in claim 33, which further includes a staging queue for the file, and which is further programmed for checking whether a previous commit is in progress after asynchronously writing to the file and before obtaining the lock for the file for committing the metadata block to the file, and upon finding that a previous commit is in progress, placing a request for committing the metadata block to the file on the staging queue for the file.

47. A network file server comprising storage for storing a file, and at least one processor coupled to the storage for providing clients with concurrent write access to the file, wherein the network file server is programmed for responding to a concurrent write request from a client by:

(a) preallocating a block for the file; and then

(b) asynchronously writing to the file; and then

(c) committing the block to the file;

wherein the network file server includes a partial block conflict queue for indicating a concurrent write to a new block that is being copied at least in part from an original block of the file, and wherein the network file server is programmed for responding to a client request for a partial write to the new block by checking the partial block conflict queue for a conflict, and upon finding an indication of a conflict, waiting until resolution of the conflict with the concurrent write to the new block of the file, and then performing the partial write to the new block of the file.

48. The network file server as claimed in claim 47, which further includes a partial write wait queue, and wherein the network file server is programmed for placing a request for the partial write in the partial write wait queue upon finding an indication of a conflict, and performing the partial write upon servicing the partial write wait queue.

49. A network file server comprising storage for storing a file, and at least one processor coupled to the storage for providing clients with concurrent write access to the file, wherein the network file server is programmed for responding to a concurrent write request from a client by:

(a) preallocating a metadata block for the file; and then

(b) asynchronously writing to the file; and then

(c) committing the metadata block to the file;

wherein the network file server is programmed for gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining a lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file.

50. The network file server as claimed in claim 49, which is further programmed for checking whether a previous commit is in progress after asynchronously writing to the file and before obtaining the lock for the file for committing the metadata block to the file, and upon finding that a previous commit is in progress, placing a request for committing the metadata block to the file on a staging queue for the file.

51. A network file server comprising disk storage containing a file system, and a file system cache storing data of blocks of a file in the file system, wherein the network file server is programmed for responding to a concurrent write request from a client by:

(a) preallocating a metadata block for the file; and then

(b) asynchronously writing to the file; and then

(c) committing the metadata block to the file;

wherein the network file server is further programmed for responding to concurrent write requests by writing new data for specified blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache, and

wherein the network file server is programmed for responding to concurrent read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing to the file system cache a file block that has become stale.

52. The network file server as claimed in claim 51, which is further programmed for checking a read-in-progress flag for a file block upon finding that the file block is not in the file system cache, and upon finding that the read-in-progress flag indicates that a prior read of the file block is in progress from the file system in the disk storage, waiting for completion of the prior read of the file block, and then again checking whether the file block is in the file system cache.

53. The network file server as claimed in claim 51, which is further programmed for setting a read-in-progress flag for a file block upon finding that the file block is not in the file system cache and then beginning to read the file block from the file system in disk storage, clearing the read-in-progress flag upon writing to the file block on disk, and inspecting the read-in-progress flag to determine whether the file block has become stale due a concurrent write to the file block.

54. The network file server as claimed in claim 51, which is further programmed for maintaining a generation count for each read of a file block from the file system in the disk storage in response to a read request for a file block that is not in the file system cache, and checking whether a file block having been read from the file system in the disk storage has become stale by checking whether the generation count for the file block having been read from the file system is the same as the generation count for the last read request for the same file block.

58. A network file server comprising storage for storing a file, and at least one processor coupled to the storage for providing clients with concurrent write access to the file, wherein the network file server is programmed with a write thread for responding to a concurrent write request from a client by:

(a) obtaining an allocation mutex for the file; and then

(b) preallocating new metadata blocks that need to be allocated for writing to the file; and then

(c) releasing the allocation mutex for the file; and then

(d) issuing asynchronous write requests for writing to the file;

(e) waiting for callbacks indicating completion of the asynchronous write requests; and then

(f) obtaining the allocation mutex for the file; and then

(g) committing the preallocated metadata blocks; and then

(h) releasing the allocation mutex for the file.

59. The network file server as claimed in claim 58, which further includes an uncached write interface, a file system cache and a cached read-write interface, and wherein the uncached write interface bypasses the file system cache for sector-aligned write operations.

60. The network file server as claimed in claim 59, wherein the network file server is further programmed to invalidate cache blocks in the file system cache including sectors being written to by the uncached write interface.

61. A network file server comprising storage for storing a file, and at least one processor coupled to the storage for providing clients with concurrent write access to the file, wherein the network file server is programmed for responding to a concurrent write request from a client by:

(a) preallocating a block for writing to the file;

(b) asynchronously writing to the file; and then

(c) committing the preallocated block;

wherein the network file server also includes an uncached write interface, a file system cache, and a cached read-write interface, wherein the uncached write interface bypasses the file system cache for sector-aligned write operations, and the network file server is programmed to invalidate cache blocks in the file system cache including sectors being written to by the uncached write interface.

62. The method as claimed in claim 1, which further includes a final step of returning to said client an acknowledgement of the writing to the file.

63. The method as claimed in claim 13, which further includes a final step of returning to said client an acknowledgement of the writing to the file.

64. The method as claimed in claim 15, which further includes a final step of returning to said client an acknowledgement of the writing to the file.

65. The method as claimed in claim 25, which further includes a final step of returning to said client an acknowledgement of the writing to the file.

67. The method as claimed in claim 32, which further includes a final step of returning to said client an acknowledgement of the writing to the file.

68. The method as claimed in claim 1, which further includes a final step of saving the file in disk storage of the network file server.

69. The method as claimed in claim 13, which further includes a final step of saving the file in disk storage of the network file server.

70. The method as claimed in claim 15, which further includes a final step of saving the file in disk storage of the network file server.

71. The method as claimed in claim 25, which further includes a final step of saving the file in the disk storage.

73. The method as claimed in claim 32, which further includes a final step of saving the file in disk storage of the network file server.

EVIDENCE APPENDIX

None.

IX. RELATED PROCEEDINGS APPENDIX

None.